A TOPOGRAPHIC MAP CONCEPTUAL DATA MODEL

Martin Tuori
DCIEM
PO Box 2000
Downsview, Ontario
Canada M3M 3B9

George Moon
Collins & Moon Limited

Abstract

This paper describes the development of a conceptual data model for topographic map applications. This work is part of a larger project, studying interactive spatial information systems (ISIS). The conceptual data model is being used as the basis for a prototype spatial database management system (SDBMS). The intended applications are not in map design or production, but the use of topographic information for ad hoc queries and problem solving. Such applications arise in military command and control, urban planning, and real estate, and are characterized by the need to ask questions about geography, objects situated in a geographic context, and relationships among them. Although each application area probably requires its own data model, our data modeling approach is applicable to all of them. We argue in favour of a top-down or descriptive data model, rather than the bottom-up or synthetic approach sometimes used in computer vision and computer aided design data models.

The approach described uses the Entity-Relationship (ER) modeling approach for high level conceptual data modeling, and a relational database management system (DBMS) for physical database design and construction. The paper outlines the steps taken in developing the data model, including entity and relationship enumeration, attribute definition, name refinement and diagram construction. Converting the conceptual data model into a physical database may require some compromise, depending on the characteristics of the DBMS. We mention two DBMS features that are useful in supporting spatial applications -- a variable length bulk data type, and a mechanism for attaching external function calls to the high level query language. We include subsets of the ER diagram and relational table outlines for the topo data model, and indicate the types of questions that this data model supports. Questions can be supported in three ways -- ad hoc queries can be asked directly, using a high level query language; more complex or often repeated queries can be stored as scripts; and spatial analysis procedures can be developed to answer questions that go beyond the capability of the high level query language. We point out problems that have arisen during the evolution of the data model, and outline areas for further investigation.

Introduction

Since the 1960's there has been interest in using computer systems for spatial applications in such fields as cartography, geography, image processing, and computer aided design. Recently, there has been a movement towards the use of sophisticated DBMSs for the data storage and administration component of such systems (Blaser, 1980; Buchmann and Kunii, 1979; Chang and Fu, 1980; Hagy and Wagle, 1979; Tuori, 1982). Database management has, until recently, grown largely out of the needs of business applications; the evolution of DBMSs to handle spatial, and other less conventional applications is an encouraging development. It is not sufficient that spatial applications have a loosely structured data file in which to place graphical and alphanumeric data. The information that that data represents must be organized to support intended (and hopefully, future) applications. The database designer(s), however, are faced with another requirement -- that of choosing a structure that can be administered and supported successfully. When using the relational data model, for example, the designer should choose a design that conforms with the theory of normal forms (Kent, 1983), to minimize redundancy, and to reduce update and deletion anomalies.

Various data modeling approaches have been developed and promoted for this purpose, including the big three (hierarchical, network and relational), and more recently, the hypergraph (Bouille, 1983), semantic network, and ER (entity-relationship) approaches (Chen, 1976).

For our efforts in developing a prototype SDBMS (McCann, et al, 1984), we have chosen a relational DBMS (Rhodnius, 1983), with the initial data design being done as an ER diagram. Since the intended applications of our SDBMS are spatial planning and problem solving, our data model proceeds from a top-down point of view. This is in contrast to the bottom-up approach sometimes favoured in data models to support constructive CAD applications (Uno and Matsuka, 1980). Since the structure and naming used in the data model form the framework of the user interface, representing the end users' point of view is crucial. For example, in a data model containing information about roads, the top-down approach would contain objects called road, with attributes such as name, type, picture, etc.; the bottom-up approach, on the other hand, might view a road as a special case of an object called line-object, with attributes feature type (road in this case), name, type, picture, etc. To ask a question about roads through these two models, a person might be required to say (respectively):

show road such that name = "highway 12"

and,

show line-object such that feature-type = "road" and name = "highway 12"

We feel strongly that the structure and naming conventions used

in the data model should be selected to help people use the information contained in the system.

Using the ER Approach

The ER approach is the first step we take in designing a conceptual data model. The objective at this stage is to enumerate the classes of objects and relationships about which we expect the end user to form statements and questions. In our topographic map application, this leads easily to a list that includes roads, rivers, towns, lakes, and islands. This enumeration must also identify the significant relationships that exist between these classes of objects. In our example, this includes road-cross-river, island-in-lake, and lake-on-island. The entities and their relationships are conveniently depicted as a diagram, called an ER diagram, in which a rectangle is used to represent each class of object, and a line joining two classes indicates a relationship between those two classes. Each entity class, and each relationship class can have many instances. A subset of the ER diagram for our topo data model is shown in Figure 1.

The choice of names at this stage is important, and should be done from the end users' point of view. The terminology with which they are familiar and comfortable is more relevant than that of the system designers and implementors (if different). There is tremendous room for choice, not only in name selection, but also in conceptual framework. For example, we might choose to represent land and water features as island and lake, or as a single class, shoreline. While the latter may represent a useful geographic concept, we feel that it would unduly restrict our end users' ability to converse with the system using the familiar concepts of lakes and islands. A third possibility would be to include all three object classes; this is possible with DBMSs that support views, through which renaming and other conceptual mappings are possible.

The ER diagram should be viewed an an evolutionary design, subject to change as necessary. During this evolution, the designers can choose to simplify the diagram, by collapsing multiple classes into a single class, or to expand a class into subclasses. In our example, we could choose to replace river and lake with the single class water-feature; moving in the other direction, we could subdivide river into distinct classes for river, stream, and creek. The process of integration (aggregation) leads ultimately to a diagram consisting of a single class of all objects, and a single relationship, object-somehow-related-to-object. The process of differentiation leads ultimately to a diagram in which each real world instance of an object is a separate class. While neither extreme is useful, the processes of integration and differentiation can be applied in finding reasonable middle ground.

With the diagram at least partially in place, we can look more closely at each class of objects or relationships, in an attempt to identify the characteristics or attributes that should be expressed for each instance of that class. Our class

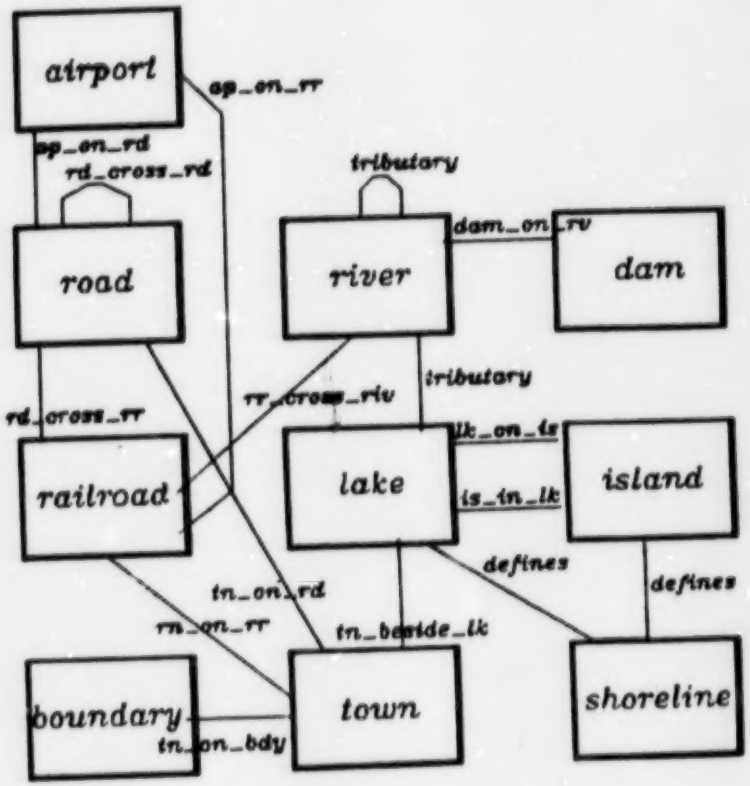

Figure 1
Topo Data Model (subset)

road, for example, could have attributes unique-id, name, type, picture, width, etc. It is useful, though not necessary, to avoid using the same word to mean different things. When we use the word type in describing a road, we mean something very different than when we use the same term in describing a river. A reference to type, without specifying the class would be ambiguous, since the domain (possible range of values) would be indeterminate. We can resolve this ambiguity at some cost in naming complexity, by introducing separate terms, road-type, and river-type.

In some cases, it is not clear whether a concept should be modeled as an object class, or as an attribute of an existing object or relationship. Bridges can be thought of as objects in their own right, or as an attribute of a crossing between other objects. Since a bridge is a complex object in its own right, we have chosen to make it into a separate class. This leaves us with the problem of relating the concepts of crossing and bridge. Not every crossing involves a bridge, so we cannot simply integrate the two classes. The ER approach, however, does not indicate how we can draw a relationship between an object class and a relationship class. Two alternatives are shown in Figure 2. In the first approach, we relate bridge to the road it supports, with no direct relationship to the crossing. Any question that relates crossings and bridges must be carried out using that implicit understanding that they are related if they occur at the same coordinates, and are both related to the same road. The second approach uses an adaptation of the ER diagram, in which relationships can be attached to other relationships.

The names that have been used to identify object classes, relationship classes, and the attributes form a data dictionary for this conceptual design. As much as possible, this lexicon should be used as the basis for the next stage, database design and construction.

Database Design and Construction

To create a relational database from the ER diagram is a straightforward process. Each object or relationship class becomes a table (relation) in the database, and the attributes listed for each class become the attributes (columns) of . The lexicon that has been created during the ER should be used as the basis for the database design, if possible. If a word chosen earlier is in conflict with a keyword of the DBMS query language, however, we would change our name to eliminate the problem.

Further integration is possible at this stage, if we want to group classes with identical attributes into a single table. In our example, the various crossing relationships could be built as a single table called crossing. Note, however, that the separate versions contain implicit information about the classes of the participants, while the integrated version does not. To implement a road-cross-river table, we would identify the road and river by the unique identification number for each, knowing that the first was a road, and the second a river. In a

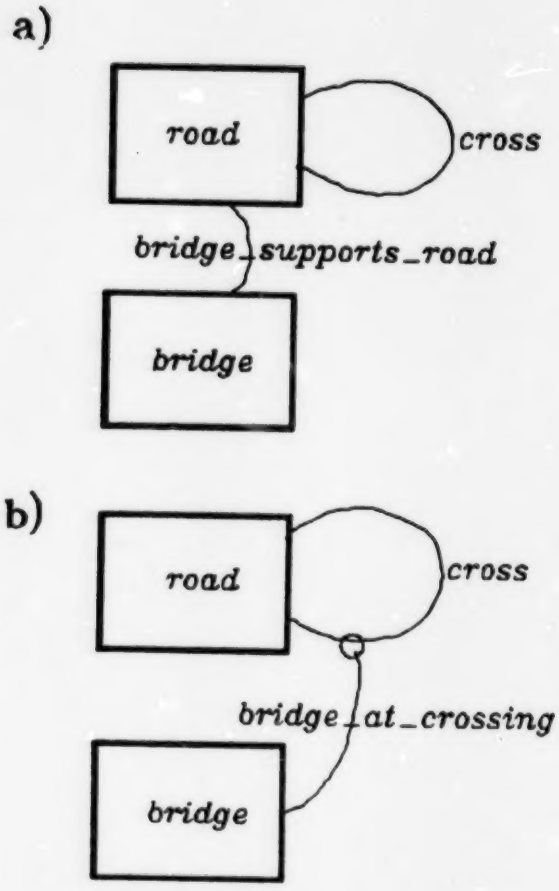

Figure 2
An Awkward Relationship

combined crossing table, however, we would also be forced to note the class of each participant in the relationship. This makes for fewer table names, at the expense of having a wider single table.

Having outlined the structure of the relational database, we must now decide the types of data that are to be stored for each attribute in each table. The choices that are typical in a DBMS are 8, 16, and 32 bit integer, floating point, character strings, dollar values, and dates. Some DBMSs offer additional data types; the version of the Mistress DBMS that we are using permits the storage and retrieval of variable length, undifferentiated bulk data. We are using this feature to store vector graphics information, and raster graphics images. It could also be used to store digitized sound samples, or any block of raw data. The data may have an internal structure understood by the application; but to the DBMS it is an undifferentiated stream of bytes.

Table outlines for some of the key entities and relationships in our data model are shown in Figure 3. This is a good point in the design process to expand the data dictionary documentation, by providing a description for each attribute of each table. This description should explain, in simple language, the meaning (semantics) of each attribute, and tell the range of values that is permissible (domain). If desired, this information can also be stored in the database, using a table of the form shown in Figure 4.

Putting the Data Model to Use

Ultimately, the worthiness of an application data model is measured by its ability to support the functions for which it has been designed. Our topographic data model should support high level questions about map objects, and other objects situated on the map (a command and control data model is under development, to mesh with the model presented here). Given the relational database we have outlined, the following queries can be posed directly to the DBMS, using a query language like SQL.

To find roads under construction, of type freeway, we could use the following query:

```
show road aspects picture, name, id such that type = 'free-
way' and condition = 'under construction';
```

This would produce a list of road names and ids (unique identification numbers), and their outlines would be drawn on a graphic display. This example involves only one table in the relational database; to illustrate the use of a relational join operation, consider asking for towns situated along a specific road:

```
show town, road, tn-on-rd aspects town.name such that tn-on-
rd.townid = town.id and tn-on-rd.roadid = road.id and
road.name = 'highway 401';
```

There is an implied join among the three tables town, road, and

road

attributes:	id	name	picture	type	condition	...
data type:	int	string	metafile	string	string	
domain:			bulk graphics	dirt gravel paved expressway ...	normal under repair abandoned destroyed	

town

id	name	x	y	picture	...
int	string	long int	long int	metafile	

lake

id	name	x	y	picture	...
int	string	long int	long int	metafile	

tn_on_rd

townid	roadid
int	int

Figure 3

Sample Table Outlines

data_dictionary

name	type	datatype	description
string	string	string	string
	table/attribute	char/int/long int/...	

Figure 4

Data Dictionary Outline

tn-on-rd, that considers combinations of tuples from these tables, that satisfy the given conditions. The syntax of the query language permits arbitrary combinations of tables, and complex conditions. In situations where the same queries are used repeatedly, they can be saved as prepared scripts, or macros, and run with argument substitution:

```
ask-tn-on-rd:

show town, road, tn-on-rd aspects town.name such that tn-on-
rd.townid = town.id and tn-on-rd.roadid = road.id and road-
name = '$1';
```

When this script is invoked -- run ask-tn-on-rd 'highway 401' -- the argument is substituted for $1, resulting in the same query as before.

Further extension and customization is possible, if the query language permits the addition of application functions. The Mistress DBMS supports this facility, and will be used in this application for queries like:

```
show town, lake aspects town.picture, town.name such that
lake.name = 'Bass Lake' and distance(town.x, town.y, lake.x,
lake.y) < 10km;
```

to find towns within a given distance of Bass Lake. In this example, the function is a two-dimensional euclidean distance between two points; but the same mechanism will be used to support point-line, point-polygon and other functions. In this way, application specific spatial analysis functions can be provided, without having to rewrite the query language.

Conclusions

We have outlined an approach for developing conceptual data models, and illustrated it in the context of a general model for topographic applications. It is clear that specialized data structures could be designed to support specific applications more efficiently; but we believe that a general conceptual data model will prove more useful, by permitting extension to a wide range of problem solving activities.

Much more work is needed in this area; the effectiveness of this data modeling approach should be compared with other approaches, both at a conceptual level, and in operational contexts. We have not yet performed normalization on the relational database described; but we have loaded the database using digital topographic data at 1:50,000 scale. The selection of appropriate human interfaces to spatial information systems is of particular concern to us, and is the primary direction of our ongoing research.

References

Blaser, A., Editor, Data Base Techniques for Pictorial Applications, 599 pages, Springer-Verlag, (1980)

Bouille, F., A Structured Expert System for Cartography Based on the HBDS, Proceedings, Auto-Carto Six, pages 202-210, (Oct, 1983)

Buchmann, A.P. and Kunii, T.L., Evolutionary Drawing Formalism in an Engineering Database Environment, Proceedings of Compsac 79, pages 732-737, (Nov, 1979)

Chang, N.S. and Fu, K.S., A Query Language for Relational Image Database Systems, pages 68-73, in IEEE Workshop on Picture Data Description and Management, 1980, 305 pages, (Aug, 1980)

Chang, S.K. and Fu, K.S., Editors, Pictorial Information Systems, 445 pages, Springer-Verlag, (1980)

Chen, P.P., The Entity-Relationship Model: Towards a Unified View of Data, ACM TODS, 1, 1, pages 9-36, (Mar, 1976)

Kent, W., A Simple Guide to Five Normal Forms in Relational Database Theory, Communications of the ACM, 26, 2, pages 120-125, (Feb, 1983)

Nagy, G. and Wagle, S., Geographic Data Processing, ACM Computing Surveys, 11, 2, pages 139-181, (Jun, 1979)

Rhodnius Inc., Mistress: Relational Database Management System, (1983), Toronto, Canada

Tuori, M., Spatial Information Management -- A Survey, Proceedings, Graphics Interface 82, pages 243-250, (May, 1982)

Uno, S. and Matsuka, H., A Relational Database for Design Aids System, pages 89-94, in IEEE Workshop on Picture Data Description and Management, 1980, 305 pages, (Aug, 1980)

POOR

QUALITY

ORIGINAL